Journal of Digital Imaging

Toolkits and Libraries for Deep Learning

--Manuscript Draft--

Manuscript Number:	JDIM-D-17-00010
Full Title:	Toolkits and Libraries for Deep Learning
Article Type:	Review
Section/Category:	Commentary
Funding Information:	National Cancer Institute (CA160045): Dr. Bradley Erickson, Dr. Panagiotis Korfiatis, Dr. Zeynettin Akkus, Dr. Kenneth Philbrick National Institute of Diabetes and Digestive and Kidney Diseases (DK090728): Dr. Bradley Erickson, Dr. Timothy Kline
Abstract:	Deep Learning is an important new area of machine learning which encompasses a wide range of neural network architectures designed to complete various tasks. In the medical imaging domain, example tasks include organ segmentation, lesion detection, and tumor classification. The most popular network architecture for deep learning for images is the convolutional neural network (CNN). Whereas traditional machine learning requires determination and calculation of features from which the algorithm learns, deep learning approaches learn the important features as well as the proper weighting of those features to make predictions for new data. In this paper, we will describe some of the libraries and tools that are available to aid in the construction and efficient execution of deep learning as applied to medical images.
Corresponding Author:	Bradley Erickson, M.D. Ph.D. Mayo Clinic - Rochester Rochester, Minnesota UNITED STATES
Corresponding Author Secondary Information:	
Corresponding Author's Institution:	Mayo Clinic - Rochester
Corresponding Author's Secondary Institution:	
First Author:	Bradley Erickson, M.D. Ph.D.
First Author Secondary Information:	
Order of Authors:	Bradley Erickson, M.D. Ph.D.
	Panagiotis Korfiatis, PhD
	Zeynettin Akkus, PhD
	Timothy Kline, PhD
	Kenneth Philbrick, PhD
Order of Authors Secondary Information:	
Author Comments:	White paper / review paper related to CMIMI conference.
Suggested Reviewers:	

Toolkits and Libraries for Deep Learning

Introduction

Deep Learning is an important new area of machine learning which encompasses a wide range of neural network architectures designed to complete various tasks(1–4). In the medical imaging domain, example tasks include organ segmentation, lesion detection, and tumor classification(5–8). The most popular network architecture for deep learning for images is the convolutional neural network (CNN). Whereas traditional machine learning requires determination and calculation of features from which the algorithm learns, deep learning approaches learn the important features as well as the proper weighting of those features to make predictions for new data. In this paper, we will describe some of the libraries and tools that are available to aid in the construction and efficient execution of deep learning as applied to medical images.

How to Evaluate a Toolkit

There is not a single criterion for determining the best toolkit for deep learning. Each toolkit was designed and built to address the needs perceived by the developer(s) and also reflects their skills and approaches to problems. Therefore in this report, we will attempt to objectively assess each toolkit using a range of different criteria, but in many cases, the assessment is subjective. Therefore, this is mostly a description of the tools. From this, it is hoped that readers can determine the toolkits that are most likely to work well for them.

The criteria we describe include:

Language: The computer language in which the toolkit was written. This can be important if you expect you will adjust some parts of the toolkit internals. The language will also impact to at least some degree the language(s) you can use for your development, though many of the toolkits do have bindings or other mechanisms that allow you to access the toolkit from a language different from what it was written in.

Documentation: The availability of documentation as well the availability of examples. Clearly, high quality documentation as well as examples that are similar to problems you will work on will be helpful to efficiently developing a solution to your problem. Good documentation is also a sign that the tool is mature and not changing rapidly.

Development Environment: The ease of programming to create a network. This is a highly subjective assessment, and we have tried to identify objective properties that will affect a person's subjective evaluation. For example, some toolkits have graphical integrated development environment. Some will prefer this while others prefer a dedicated editor and command line. Some have visualization tools for affirming the network is correct; others have visualization tools for monitoring the learning progress.

Execution Speed: This is the speed of actually classifying or segmenting the image using a trained network. While it will involve hundreds to thousands of calculations per pixel, this can usually be accomplished in seconds for medical uses. Usually this is substantially faster than training and is usually much less important than training speed for medical applications.

Training Speed: While execution of a trained network can be important in some cases, the training time is usually many orders of magnitude slower. Therefore, training speed is likely to be of greater consequence than execution speed. The training speed depends on how efficient the math libraries are and how well those libraries take advantage of the available computational resources, and also depends heavily on the nature of the task and images. For instance, the memory available to the processor and the bandwidth from storage to processing unit will have a huge effect on performance when training large datasets, especially when all data must be accessed at each iteration. As such, estimation of speed will be a very rough estimate.

GPU Support: Graphical Processing Units or GPUs can significantly increase the rate at which networks learn. Special libraries like cuDNN are an example of how the special type of calculations required for deep learning have been adapted and optimized for computation by a GPU. Most toolkits leverage cuDNN as their way to support GPUs. Some are able to support multiple GPUs with little developer effort, some require more effort, and some simply can't support more than 1 GPU. Supporting more than 1 GPU will often result in substantial performance gains that nearly match the number of cards added. As noted above, the coupling of storage with processing will have a large impact on performance.

Maturity Level: This is our subjective estimate of how mature a toolkit is. We estimated this using a combination of a large user base, few bug fixes in the last few months, and a good support community.

Model Library: In some cases, toolkits also have a library of code that creates networks, and may even have the weights associated with the nodes. One of the most recognized of these is the Caffe 'Model Zoo', where one may download many of the popular reference networks and weights.

Github Commits: This is the number of changes made to the toolkit code since the project was placed on github. This is an objective measure, but can be misleading, as a toolkit may have been developed using some other mechanism, and only recently placed on github. This will result in a low number of commits compared to its true maturity. Conversely, if a toolkit was started in a very early stage or with many bugs, there can be many commits despite a low level of maturity.

Github Contributors: This reflects how many different people are contributing to the project on github. In general, more contributors reflect a vibrant community with many users and also likely a broader range of users/applications.

Toolkits

Caffe

Caffe is one of the most mature toolkits, and was developed by Berkeley Vision & Learning Center. It is Modular and fast and supports multiple GPUs with little extra effort. It uses a JSON-like text file to describe the network architecture as well as the solver methods. Also has model Zoo, which is a website where you can download caffe models as well as network weights. This can help you get going very quickly with examples. However, tuning hyperparameters is more tedious than other toolkits, in part because a different solver and model file needs to be separately defined for each set of hyperparameters. Figure 1. Provides a snippet of code for the LeNet CNN architecture. The model consists of a 7-layer convolutional network consisting of convolution maxpooling and activation layer.

```
def lenet(lmdb, batch_size):
	n = caffe.NetSpec()
	n.data, n.label = L.Data(batch_size=batch_size, backend=P.Data.LMDB,
		source=lmdb, transform_param = dict(scale=1./255),ntop=2)
	n.conv1 = L.Convolution(n.data, kernel_size=5, num_output=20,
		weight_filler=dict(type='xavier'))
```

```
n.pool1 = L.Pooling(n.conv1, kernel_size=2, stride=2, pool=P.Pooling.MAX)
n.conv2 = L.Convolution(n.pool1, kernel_size=5, num_output=50,
       weight_filler=dict(type='xavier'))
n.pool2 = L.Pooling(n.conv2, kernel_size=2, stride=2, pool=P.Pooling.MAX)
n.fc1 = L.InnerProduct(n.pool2, num_output=500,
      weight_filler=dict(type='xavier'))
n.relu1 = L.ReLU(n.fc1, in_place=True)
n.score = L.InnerProduct(n.relu1, num_output=10,
      weight_filler=dict(type='xavier'))
n.loss = L.SoftmaxWithLoss(n.score, n.label)
return n.to_proto()
```

Figure 1. Example code implementing LeNet CNN written in Caffe

DeepLearning4j

DeepLearning4j is a multi-platform toolkit with GPU support developed by Andrej Karpathy and written in Java with a Scala API. This is also a mature toolkit (written in Lua) with many examples available on the internet. This is not heavily used in medical imaging, and use appears to be declining in the medical field. It has good performance and supports multiple GPUs.

Tensorflow

Tensorflow is a rather new library (at least to public view) that was developed by Google, but already has strong adoption. Performance is good, and supports multiple GPUs and CPUs. Some view it as more difficult to use directly, but tools are addressing this challenge. Tensorflow provides tools for tuning a network and monitoring performance like Tensorboard. It also has an educational tool available as a web app. (http://playground.tensorflow.org/)

Theano

Theano is a tool for creating networks using symbolic logic, and is written in Python, but takes advantage of the efficient code base of numpy, which improves performance over standard python. The symbolic approach may be a challenge for some to learn, but Theano is good for building networks, but more challenging to create complete solutions. Theano includes computation of the gradients used in learning as a 'free' byproduct of net creation, which may be useful for those wishing to focus more on network architecture than gradient computations.

Documentation quality is fair.

Keras

Keras is a library written in Python that utilizes as backend either Theano or Tensorflow. It is easier to build complete solutions, and is easy to read, in that each line of code creates 1 layer of a network. This toolkit seems to have the greatest selection of state-of-the-art algorithms (optimizers, normalization routines, activation functions). Although Keras supports both theano and tensorflow back ends the assumption for the dimension of the input data is different so careful design is needed in order for the code to be able to work using both backends. The project is well documented while a set of examples aiming at a wide variety of problems is provided. Pretrained models of commonly used architectures for transfer learning implementation are also provided. At the time of this writing, it was announced that Tensorflow would be adopting Keras as a preferred high-level package. This is not surprising, given that the Keras author--Francois Chollet--is a Google software engineer.

```
def lenet():
       model = Sequential()
       model.add (Convolution2D(6,5,5,border_mode='valid',input_shape=(1,28,28)))
       model.add (MaxPooling2D(pool_size=(2,2)))
       model.add (Activtion("sigmoid"))

       model.add (Convolution2D(16,5,5,border_mode='valid'))
       model.add (MaxPooling2D(pool_size=(2,2)))
       model.add (Activtion("sigmoid"))
       model.add(Dropout(0.5))

       model.add (Convolution2D(120,1,1,border_mode='valid'))

       model.add(Flatten())
       model.add(Dense(84))
       model.add(Activation("sigmoid"))
       model.add(Dense(10))
       model.add(Activation("softmax"))
```

Figure 2. Example code implementing LeNet CNN written in Keras

mxNet

mxNet is a deep learning framework written in C++ with many language bindings, and supports distributed computing, including multi-GPU. It provides access to both lower-level constructs as well as higher/symbolic level API. Performance is considered to be on par with other good systems, including tensorflow, caffe, etc. A number of tutorials and training examples are available on github and it has a ‘model zoo’, which is a collection networks that have been trained on various problems.

Lasagne

Lasagne is written in Python and is built on top of Theano. It is essentially a thin wrapper to make building networks easier than using Theano directly. As such, its performance largely reflects the underlying performance of Theano.

Cognitive Network Toolkit (CNTK)

CNTK is developed by Microsoft, and is described as “Visual Studio” for Machine learning. For those that have used Visual Studio for programming, this may be a gentler and more efficient way to get into deep learning. Performance is generally good. It is a rather recent addition to the publicly available toolkits, and usage is currently less than many others.

DIGITS

DIGITS was developed by NVidia, and is a web-based tool for developing deep networks. In many ways, it is like Caffe, and uses a text file not a programming language, to describe the network and parameters. It has a network visualization tool so errors in the text file are more easily identified. In addition, it has tools for visualizing the learning process and has multiple GPU support.

Torch

Torch is a mature toolkit for machine learning that is written in C. It has good documentation and can be tailored to address specific needs. Because it is written in C, performance is very good.

Pylearn2

Pylearn2 is a machine learning research library developed by Laboratoire d'Informatique des Systèmes Adaptatifs (LISA) at University of Montreal(9)**.** Pylearn2 offers a collection of classical machine learning algorithms as well as deep neural network algorithms written in python. However, Pylearn2 is not as complete other toolkits such as keras or mxnet.

Chainer

Chainer is a bit different from other toolkits because it builds the network as part of its computation. Its authors describe it that most tools are 'Define-then-run' which means you define the architecture and then run it. Chainer attempts to build and optimize its architecture as part of the learning process, or as they call it "Define-by-Run." Chainer stores its computations rather than the programming logic. This allows it to fully leverage the power of Python.

PyTorch

PyTorch is very recent entry--it was released during the writing of this manuscript. It is a python front end to the Torch computational engine. This should provide the high performance of Torch with good GPU support with a friendlier python front end. The distinction the authors claim is that this this more than a wrapper--that there is deep integration to keep points that can allow more flexibility in how the networks are constructed.

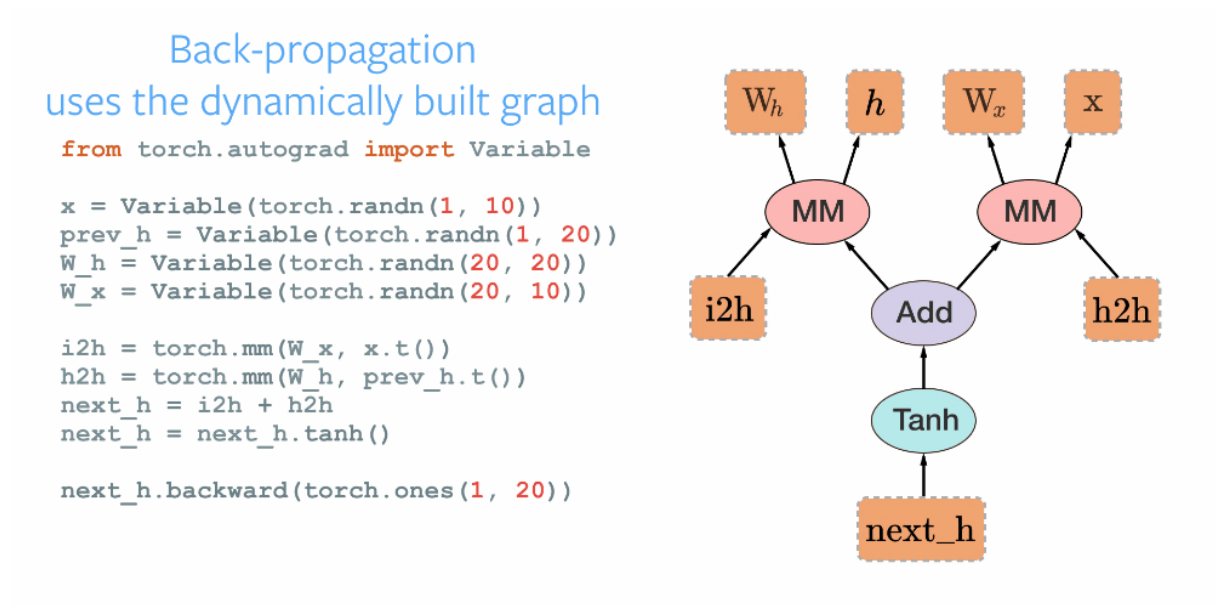

Figure 3. Example of PyTorch Code and block diagram equivalent

Other libraries

Besides the abovementioned libraries that a more broadly utilized there are more open source solutions that focus on more specific tasks. For instance Nolearn offers a good implementation of deep belief networks. Sklearn-theano offers a programming syntax that matches the one of scikit-learn (that is the main library for machine learning in Python) to work with the Theano library. Paddle is offering better capabilities for natural language processing, while h2o solutions are oriented to big data analytics offering solutions that combine well with software solutions like Spark.

Table 1. Captures the ranking of the open software libraries based on the stars and forks received by the community on github, an online repository for open source projects.

Framework	**Stars**	**Forks**	**Contributors**	**Language**
caffe	15,057	9,338	222	C++
keras	10,875	10,875	327	Python
mxnet	7,471	2,764	250	C++
torch	6,163	1,793	113	lua
convnetjs	6,128	1,198	15	javascript
deeplearning4j	5,090	1,970	103	Java
TensorFLow	4,505	667	573	Python
Paddle	4,069	1,024	53	C++
DSSTNE	3,531	559	22	C++
Chainer	1,983	512	96	Python
DIGITS	1,800	1,052	34	Python
h20	1,628	714	70	java

Table 1. Open source projects ranked based on the stars and forks received by the users.

Comparison of Toolkits

There are a few open efforts to provide benchmarks that compare the performance of these tools. One example can be found at https://github.com/soumith/convnet-benchmarks. This site compares several toolkits with several different CNN-style deep learning networks, including: AlexNet, GoogleNet and OxfordNet on a specific set of test hardware. Based on their results, Torch is faster than both Tensorflow and Caffe.

Although the majority of the deep neural network libraries are well supported by the online community, not all the libraries support multiple GPUs. The available solutions support parallelization of the computation in multiple GPUs, however the limiting factor is that the GPUs have to be in the same workstation. Limited support also exists for solutions that can parallelize the computations among different servers (for instance mxnet).

The majority of the libraries support GPU or CPU execution of the code with the CPU solutions being significantly slower. NVIDIA offers a series of GPU cards that support the necessary libraries for deep learning. For rather small neural networks, GPU cards offering 6 GB of RAM are adequate. However, as the models get larger for instance UNET (5) or RESNET (10) the memory requirements significantly increase and GPU cards with 12 or 24Gb RAM should be considered.Currently, the solutions available cover a range of users, from novice to experienced. Tools like NVIDIA Digits and deeplearning4j offer good solutions for beginners interested in exploring deep neural networks, suitable for training and educational purposes. On the other hand, libraries like Theano, Tensorflow and Torch are more appropriate for experienced users who need to have much more control over network architectures. Fortunately, there are libraries that cover the most widely used programming languages.

The open communities behind these libraries offer a wide variety of examples making application of deep neural network models easier. Additionally, Docker (http://docker.com) based solutions with all the necessary tools are provided for almost all the libraries.

Conclusions

Writing a deep learning algorithm ‘from scratch’ is probably beyond the skillset of most medical imaging researchers. It is much more efficient to utilize the tremendous resources available in a deep learning toolkit. There are many deep learning toolkits available, and we have described many in this paper. Selecting the best toolkit will depend on the skills and background of the researcher, and may also be impacted by the project and available resources. As such, it is worth spending some time to evaluate available toolkits when a project is begun, to be sure that the best one is chosen for the situation.

Acknowledgment

Supported by the National Cancer Institute U01 CA160045 and NIDDK P30 DK090728.

References

1. LeCun Y, Bengio Y. Convolutional networks for images, speech, and time series. The handbook of brain theory and neural networks. 1995;3361:1995.

2. Hinton GE, Osindero S, Teh Y-W. A fast learning algorithm for deep belief nets. Neural Comput.MIT Press; 2006;18:1527–1554.

3. Krizhevsky A, Sutskever I, Hinton GE. ImageNet Classification with Deep Convolutional Neural Networks. In: Pereira F, Burges C, Bottou L, Weinberger. KQ, editors.Advances in Neural Information Processing Systems 25. 2012. https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

4. Srivastava N, Hinton GR, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research. 2014;15:1929–1958.

5. Ronneberger O, Fischer P, Brox T. U-Net: Convolutional Networks for Biomedical Image Segmentation. In: Navab N, Hornegger J, Wells WM, Frangi AF, editors.Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015.Springer International Publishing; 2015. p. 234–241.

6. Anthimopoulos M, Christodoulidis S, Ebner L, Christe A, Mougiakakou S. Lung Pattern Classification for Interstitial Lung Diseases Using a Deep Convolutional Neural Network. IEEE Trans Med Imaging. 2016; http://dx.doi.org/10.1109/TMI.2016.2535865

7. van Grinsven M, van Ginneken B, Hoyng C, Theelen T, Sanchez C. Fast convolutional neural network training using selective data sampling: Application to hemorrhage detection in color fundus images. IEEE Trans Med Imaging. 2016; http://dx.doi.org/10.1109/TMI.2016.2526689

8. Wang S, Summers RM. Machine learning and radiology. Med Image Anal. 2012;16:933–951.

9. Goodfellow IJ, Warde-Farley D, Lamblin P, et al. Pylearn2: a machine learning research library. arXiv [stat.ML]. 2013. http://arxiv.org/abs/1308.4214

10. Szegedy C, Ioffe S, Vanhoucke V, Alemi A. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. arXiv [cs.CV]. 2016. http://arxiv.org/abs/1602.07261

Toolkits and Libraries for Deep Learning

Authors: Bradley J Erickson@
Panagiotis Korfiatis
Zeynettin Akkus
Timothy Kline
Kenneth Philbrick

Department of Radiology, Mayo Clinic, Rochester, MN, 55905
@ Corresponding Author
Email: bje@Mayo.edu